Ledesma Ayala Angel Yeremi

Git y GitHub

Git Stash:

git stash es un comando en Git que permite a los desarrolladores guardar temporalmente los cambios locales en su directorio de trabajo sin comprometerlos en un commit. Esta característica resulta especialmente útil cuando se está trabajando en una rama y se necesita cambiar a otra rama o realizar otras operaciones sin confirmar los cambios actuales.

El comando git stash realiza las siguientes acciones:

1. Guarda los cambios no confirmados en una pila (stash) temporal.
2. Deja el directorio de trabajo limpio para permitir cambios de rama u otras operaciones sin conflictos.
3. Los cambios guardados en el stash pueden ser recuperados en cualquier momento y aplicados nuevamente a la rama de trabajo.

Ejemplos de uso de git stash:

1. Guardar cambios en el stash:

   ```
   git stash save "Trabajo en progreso"
   ```

   Esto guarda los cambios locales en el stash con un mensaje descriptivo.

2. Listar los stashes:

   ```
   git stash list
   ```

   Muestra una lista de los stashes disponibles en el repositorio, numerados automáticamente.

3. Recuperar los cambios del stash:

   ```
   git stash apply stash@{0}
   ```

   Recupera los cambios guardados en el stash más reciente.

4. Recuperar y eliminar el stash:

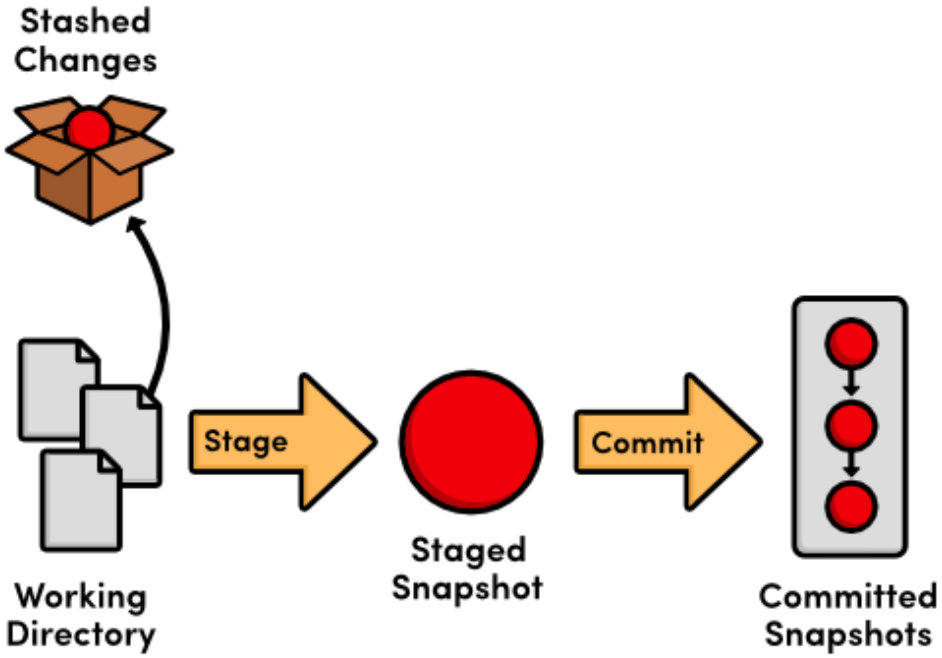
   ```
   git stash pop stash@{0}
   ```

   Recupera los cambios del stash más reciente y lo elimina de la pila.

5. Eliminar un stash específico:

   ```
   git stash drop stash@{0}
   ```

   Elimina un stash particular de la pila sin aplicar los cambios.



Git clean:

El comando git clean en Git es utilizado para eliminar archivos no rastreados en el directorio de trabajo. Los archivos no rastreados son aquellos que no están bajo control de versiones y que Git no está siguiendo. Es importante tener cuidado al usar este comando, ya que eliminará permanentemente los archivos no rastreados, y no se pueden recuperar a través de Git.

Uso básico:

```
git clean [opciones]
```

Opciones comunes:

- -n o --dry-run: Muestra los archivos que serían eliminados sin realmente eliminarlos. Es útil para verificar antes de ejecutar la limpieza.
- -f o --force: Realiza la eliminación de archivos sin pedir confirmación, una vez que estás seguro de que deseas eliminar los archivos.
- -i o --interactive: Proporciona una interfaz interactiva que permite al usuario elegir qué archivos eliminar y cuáles retener.
- -d o --directories: Además de archivos no rastreados, elimina directorios no rastreados.

Ejemplos de uso:

1. Mostrar archivos que serían eliminados:

   ```
   git clean -n
   ```

   Esto mostrará una lista de archivos y directorios que serían eliminados, pero no realizará la acción real.

2. Eliminar archivos no rastreados:

   ```
   git clean -f
   ```

   Esto eliminará permanentemente los archivos no rastreados en el directorio de trabajo. Se recomienda usar -n primero para asegurarse de lo que se eliminará.

3. Eliminar archivos y directorios no rastreados de manera interactiva:

   ```
   git clean -i
   ```

   Esto proporcionará una interfaz interactiva que te permitirá elegir qué archivos y directorios no rastreados deseas eliminar.

4. Eliminar archivos y directorios no rastreados, incluyendo directorios vacíos:

   ```
   git clean -fd
   ```

   Esto eliminará archivos y directorios no rastreados, y también eliminará directorios vacíos.

Consideraciones importantes:

- Es necesario asegurarse de revisar cuidadosamente qué archivos se eliminarán antes de ejecutar git clean, especialmente si se está utilizando la opción -f para forzar la eliminación.
- Los archivos rastreados (agregados al repositorio) no se eliminarán con git clean, ya que Git los considera bajo su gestión.
- git clean no afecta los archivos en el área de preparación (staging area) ni los commits realizados. Solo elimina archivos y directorios no rastreados en el directorio de trabajo.

Este comando es útil para limpiar el directorio de trabajo de archivos y directorios generados automáticamente o descargados durante el desarrollo, dejándolo en un estado más limpio y manejable.

Ledesma Ayala Angel Yeremi

Cherry-pick:

git cherry-pick es un comando de Git que permite aplicar un commit específico de una rama a otra. Esto es útil cuando se desea traer cambios específicos de una rama a otra sin fusionar toda la rama. Esencialmente, se están "seleccionando cerezas" o commits específicos de una rama y aplicándolos en otra.

Uso básico:

```
git cherry-pick <commit-hash>
```

Ejemplo:

Supongamos que hay dos ramas, rama-origen y rama-destino. Para aplicar un commit específico de rama-origen a rama-destino, primero, se obtiene el hash del commit que se desea aplicar y luego se ejecuta:

```
git checkout rama-destino git cherry-pick <commit-hash>
```

Opciones comunes:

- -e o --edit: Abre el mensaje del commit en un editor de texto para que puedas modificarlo antes de realizar el cherry-pick.
- -n o --no-commit: Realiza todas las operaciones necesarias, pero no realiza el commit final. Esto te da la oportunidad de hacer ajustes antes de confirmar el cherry-pick.
- -x: Agrega una línea en el mensaje del commit que indica que ha sido cherry-picked.

Ejemplos de uso:

1. Cherry-pick de un commit específico:

   ```
   git checkout rama-destino git cherry-pick <commit-hash>
   ```

   Esto aplicará el commit identificado por <commit-hash> de rama-origen en la rama actual (rama-destino).

2. Cherry-pick con edición del mensaje del commit:

   ```
   git cherry-pick -e <commit-hash>
   ```

   Esto abrirá el mensaje del commit en un editor para que se puedan realizar cambios antes de confirmar el cherry-pick.

3. Cherry-pick sin realizar el commit final:

   ```
   git cherry-pick -n <commit-hash>
   ```

Esto realiza todas las operaciones necesarias para el cherry-pick, pero no realiza el commit final, permitiéndote hacer ajustes antes de confirmar.

4. Cherry-pick con indicación de que fue cherry-picked:

   ```
   git cherry-pick -x <commit-hash>
   ```

   Agrega una línea al mensaje del commit que indica que ha sido cherry-picked.

Consideraciones importantes:

- Pueden surgir conflictos durante el cherry-pick, especialmente si el commit que se está aplicando modifica las mismas líneas de código que han cambiado en la rama de destino desde el commit original.
- Después de realizar el cherry-pick, es posible que se desee realizar un commit para confirmar los cambios en la rama de destino.
- Se deben evitar cherry-picks excesivos, ya que esto puede complicar el historial de Git si no se manejan correctamente.

El comando git cherry-pick es una herramienta poderosa para seleccionar y aplicar cambios específicos entre ramas, lo que facilita la gestión selectiva de commits en un proyecto.